Accelerating Your Application I/O on HPC Systems

Poster Presentation

August 18, 2020 Cameron Stanavige





Accelerating Your Application I/O on HPC Systems

Kathryn Mohror (PI), Adam Moody, Elsa Gonsiorowski, Cameron Stanavige, Tony Hutter (Lawrence Livermore National Laboratory) Sarp Oral (Co-PI), Feiyi Wang, Hyogi Sim, Mike Brim, Swen Boehm (Oak Ridge National Laboratory) Craig Steffen, Celso Mendes (National Center for Supercomputing Applications)



Abstract

As the HPC community pushes towards exascale, HPC clusters are becoming increasingly complex. Part of this complexity is introduced by the addition of new storage tiers, such as burst buffers. Each tier may have different performance characteristics and may be subject to different policies regarding availability or allocation. A number of tools are under active development to improve file set management, both between tiers and within a single tier.

Types of Burst Buffers

1	CN	CN	CN	CN		CN	CN	CN	CN	CN
4	014	014	014	014		CIV	CIT	CIT	OIV	Oi
	CN	CN	CN	CN		CN	CN	CN	CN	CN
	CN	CN	CN	CN		CN	CN	CN	CN	CN
	CN	CN	CN	CN			ION	ION	ION	_
	Ť		ľ		Т	т	一			
arallel File System						Parallel File System				

Compute Node Namespace Global Namespace
Single Allocation Multiple Allocations

MPIFileUtils

https://github.com/HPC/mpifileutils

Available Tools Help wanted! Suite of MPI-based file utilities to manage large datasets, including large directory trees and large files. ➤ dbcast dgre dbz2 dparallel ➤ dchmod ➤ dsh ➤ dtar ➤ dcmp dcmp 60× Collaboration ➤ dcp Speedup compared to regular Unix utility, using 256 MPI ➤ ddup > LANL processes, operating on 1 million files. ➤ dfind ➤ LLNL Workflows ➤ ORNL ➤ drein Data Migration dwalk -> dcp -> dcmp --lite -> dsync ➤ drm ➤ DDN ➤ dstripe ➤ RedHat ➤ dsync ➤ FSU LC Policy Implementation mpirun -np 128 drm --aggressive ./_AutoDelete ➤ dwalk > ANU

SCR: Scalable Checkpoint Restart Library

Allows MPI applications to utilize hierarchical storage for high I/O bandwidth.

CN

CN

- ➤ Simple integration by marking application I/O
- Support for file-per-node I/O patterns
- > Automatic management of checkpoint vs. output

Usage

SCR_Start_output("dataset name", flags); SCR Route file(path, newpath);

int rc = MyApp_Checkpoint(newpath);

SCR_Complete_output(rc);

Job Management Scripts

- > Reliability through cross-failure domain redundancy
- ➤ Automatic hang-detection and restart
- > Scavenge data from down nodes
- > Support for burst buffer asynchronous post-stage

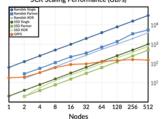
Configurations

LANL Trinity, NERSC Cor

- > Built-in support for various resource managers and burst buffers
- Leverage multiple asynchronous I/O technologies

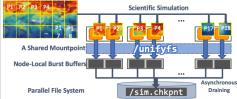
https://github.com/LLNL/SCR

SCR Scaling Performance (GB/s)



UnifyFS

Creates a Shared Namespace on Node-Local Storage



UnifyFS: Project Description and Scope

- User-level file system
- > Highly-specialized for shared file access on HPC systems with distributed, node-local burst buffers
- Integration with resource managers to instantiate UnifyFS in user jobs

Will Support Common HPC I/O Use Cases

- Disjointed write and read phases, ideal for most checkpoint workload
- > Scientific applications generating periodic output data
- Ensemble applications sharing data through files
- ➤ Will support HPC I/O libraries such as HDF5, ADIOS, MPI-I/O, PnetCDF

Using UnifyFS in a Job is Easy #!/bin/bash -1

#SBATCH -N 1024

xport UNIFYFS_LOGIO_SPILL_DIR=/mnt/ssd/\$USER/data nifyfs start --mount=/unifyfs --stage-in=/pfs/data

run -n 4096 ./simulation

nifyfs terminate --stage-out=/pfs/data/out

Simply Change File Path void checkpoint(void) { int rank;

https://github.com/LLNL/UnifyFS

MPI Comm rank(MPI COMM WORLD, &rank);

-// file = "/pfs/shared.chpt";
file = "/unifyfs/shared.ckpt";

File *fs = fopen(file, "w");



- > Results on Summit show scalable write performance for UnifyFS with shared files on burst buffers
- ➤ Recent optimizations** can speed up reads by up to 236x over our baseline ** in the case where a process only reads in the bytes it wrote





Lawrence Livermore





Accelerating Your Application I/O on HPC Systems

Kathryn Mohror (PI), Adam Moody, Elsa Gonsiorowski, Cameron Stanavige, Tony Hutter (Lawrence Livermore National Laboratory)

Sarp Oral (Co-PI), Feiyi Wang, Hyogi Sim, Mike Brim, Swen Boehm (Oak Ridge National Laboratory)

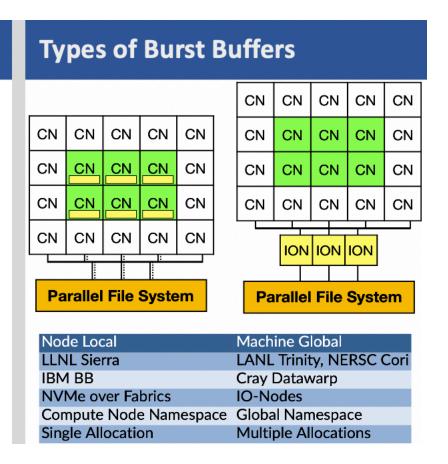
Craig Steffen, Celso Mendes (National Center for Supercomputing Applications)





Abstract

As the HPC community pushes towards exascale, HPC clusters are becoming increasingly complex. Part of this complexity is introduced by the addition of new storage tiers, such as burst buffers. Each tier may have different performance characteristics and may be subject to different policies regarding availability or allocation. A number of tools are under active development to improve file set management, both between tiers and within a single tier.



MPIFileUtils

https://github.com/HPC/mpifileutils

Suite of MPI-based file utilities to manage large **Available** datasets, including large directory trees and large files. > dbcast

Tool	Speedup
dcp	51×
dcmp	60×
drm	5×

Speedup compared to regular Unix utility, using 256 MPI processes, operating on 1 million files.

Workflows

Data Migration
dwalk -> dcp -> dcmp --lite -> dsync

LC Policy Implementation
mpirun -np 128 drm --aggressive ./_AutoDelete

Available Tools Help wanted!

- ➤ dbcast
 ➤ dgre
- ➤ dbz2
 ➤ dparallel
- ➤ dchmod
 ➤ dsh
- ➤ dcmp
 ➤ dtar
- ➤ dcp Collaboration
- ➤ ddup
 ➤ LANL
- ➤ dfind
 ➤ LLNL
- ➤ dreln
 ➤ ORNL
- ➤ drm
 ➤ DDN
- ➢ dstripe
 ➢ RedHat
- ➤ dsync
 ➤ FSU
- ➤ dwalk
 ➤ ANU

Suite of MPI-based file utilities to manage large datasets, including large directory trees and large files.

Tool	Speedup
dcp	51×
dcmp	60×
drm	5×

Speedup compared to regular Unix utility, using 256 MPI processes, operating on 1 million files.

```
Workflows
Data Migration
dwalk -> dcp -> dcmp --lite -> dsync

LC Policy Implementation
mpirun -np 128 drm --aggressive ./_AutoDelete
```



Available Tools Help wanted!

➤ dbcast
➤ dgre

➤ dbz2
➤ dparallel

➤ dchmod
➤ dsh

➤ dcmp
➤ dtar

➢ dcp Collaboration

➤ ddup
➤ LANL

➤ dfind
➤ LLNL

➤ dreln
➤ ORNL

 \triangleright drm \triangleright DDN

➤ dstripe
➤ RedHat

➤ dsync
➤ FSU

➤ dwalk
➤ ANU



SCR: Scalable Checkpoint Restart Library

Allows MPI applications to utilize hierarchical storage for high I/O bandwidth.

Library

- Simple integration by marking application I/O phases
- ➤ Support for file-per-node I/O patterns
- ➤ Automatic management of checkpoint vs. output files

Usage

SCR_Start_output("dataset name", flags);
SCR_Route_file(path, newpath);
int rc = MyApp Checkpoint(newpath);

SCR_Complete_output(rc);

Job Management Scripts

- Reliability through cross-failure domain redundancy
- > Automatic hang-detection and restart
- > Scavenge data from down nodes
- ➤ Support for burst buffer asynchronous post-stage

Configurations

- ➤ Built-in support for various resource managers and burst buffers
- ➤ Leverage multiple asynchronous I/O technologies

Ramdisk Single Ramdisk Partner Ramdisk XOR SSD Single SSD Single SSD XOR OFFS 10⁴ 10² 10¹ 1 2 4 8 16 32 64 128 256 512 Nodes

SCR Scaling Performance (GB/s)

https://github.com/LLNL/SCR

Library

- Simple integration by marking application I/O phases
- > Support for file-per-node I/O patterns
- ➤ Automatic management of checkpoint vs. output files

Usage

```
SCR_Start_output("dataset name", flags);
SCR_Route_file(path, newpath);
int rc = MyApp_Checkpoint(newpath);
SCR_Complete_output(rc);
```

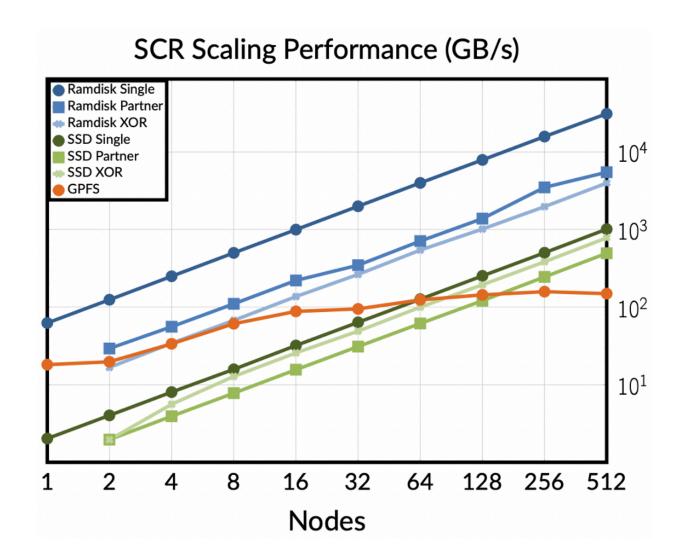


Job Management Scripts

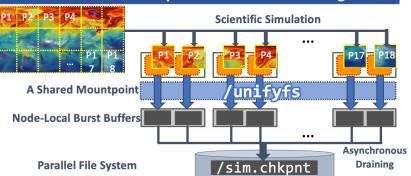
- Reliability through cross-failure domain redundancy
- Automatic hang-detection and restart
- Scavenge data from down nodes
- Support for burst buffer asynchronous post-stage

Configurations

- Built-in support for various resource managers and burst buffers
- ➤ Leverage multiple asynchronous I/O technologies



UnifyFS Creates a Shared Namespace on Node-Local Storage Usi



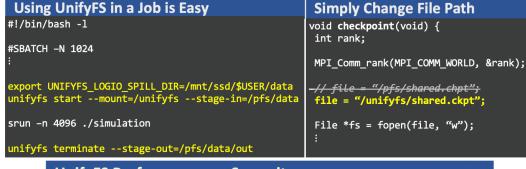
UnifyFS: Project Description and Scope

- User-level file system
- Highly-specialized for shared file access on HPC systems with distributed, node-local burst buffers
- ➤ Integration with resource managers to instantiate UnifyFS in user jobs

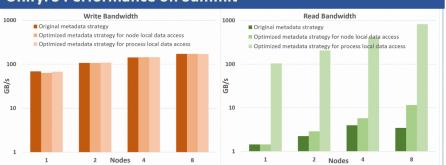
Will Support Common HPC I/O Use Cases

- > Disjointed write and read phases, ideal for most checkpoint workload
- > Scientific applications generating periodic output data
- > Ensemble applications sharing data through files
- ➤ Will support HPC I/O libraries such as HDF5, ADIOS, MPI-I/O, PnetCDF

https://github.com/LLNL/UnifyFS

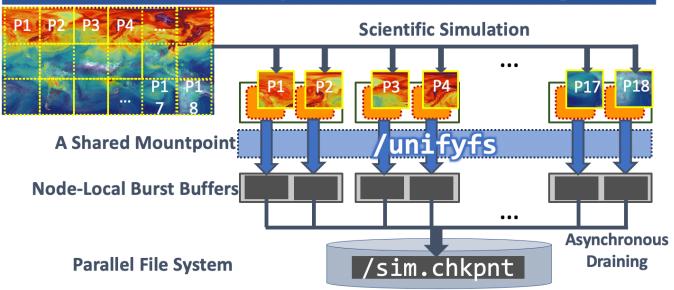


UnifyFS Performance on Summit



- ➤ Results on Summit show scalable write performance for UnifyFS with shared files on burst buffers
- Recent optimizations** can speed up reads by up to 236x over our baseline performance
 ** in the case where a process only reads in the bytes it wrote

Creates a Shared Namespace on Node-Local Storage



UnifyFS: Project Description and Scope

- ➤ User-level file system
- ➤ Highly-specialized for shared file access on HPC systems with distributed, node-local burst buffers
- > Integration with resource managers to instantiate UnifyFS in user jobs

Will Support Common HPC I/O Use Cases

- > Disjointed write and read phases, ideal for most checkpoint workload
- > Scientific applications generating periodic output data
- ➤ Ensemble applications sharing data through files
- ➤ Will support HPC I/O libraries such as HDF5, ADIOS, MPI-I/O, PnetCDF



```
#!/bin/bash -1
#SBATCH -N 1024
:
export UNIFYFS_LOGIO_SPILL_DIR=/mnt/ssd/$USER/data
unifyfs start --mount=/unifyfs --stage-in=/pfs/data
srun -n 4096 ./simulation
unifyfs terminate --stage-out=/pfs/data/out
Simply Change File Path

void checkpoint(void) {
   int rank;

MPI_Comm_rank(MPI_COMM_WORLD, &rank);

// file = "/pfs/shared.chpt";

file = "/unifyfs/shared.ckpt";

File *fs = fopen(file, "w");
   :

unifyfs terminate --stage-out=/pfs/data/out
```

UnifyFS Performance on Summit



- ➤ Results on Summit show scalable write performance for UnifyFS with shared files on burst buffers
- Recent optimizations** can speed up reads by up to 236x over our baseline
 performance
 ** in the case where a process only reads in the bytes it wrote





This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations - the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, to support the nation's exascale computing imperative.

https://github.com/HPC/mpifileutils

https://github.com/LLNL/SCR

https://github.com/LLNL/UnifyFS

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

